A PROJECT REPORT ON

TIC-TAC-TOE USING BASYS3 ARTIX-7

Submitted to Dept. of ECE

By

D.Rupesh Babu(R151737)

M.Ashok Kumar(R151690)

In partial fulfilment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Under the guidance of

Mr.Venkateshwarulu Naik

Rajiv Gandhi University of Knowledge Technologies-AP

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

DEPARTMENT OF

ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this mini Project entitled "**TIC-TAC-TOE USING BASYS3 ARTIX-7 "** submitted by **D.RUPESH BABU(R151737),M.ASHOK KUMAR(R151690).**Under the guidance of **Mr.VENKATESHWARULU NAIK** in partial fulfilment of the academic requirements for the award of degree of **Bachelor of Technology** in **Electronics and Communication Engineering** by **RGUKT,R K VALLEY** during the academic year 2019-2020.

Internal Guide

Mr.Venkateshwarulu Naik,
Assistant Professor,
Dept. of ECE,RGUKT,
RK Valley.

Head of the Department

Mr.Sreekanth Reddy,
Assistant Professor,
Dept. of ECE,RGUKT,
RK Valley.

DECLARATION

We **D.Rupesh Babu** and **M.Ashok Kumar** hereby declare that this report entitled **"TIC-TAC-TOE USING BASYS3 ARTIX-7"** submitted by us under the guidance and supervision of **Mr.Venkateshwarulu Naik** is a bonafide work. We also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma.

Date: 21-11-2019 D.Rupesh Babu(R151737)

Place:R K Valley M.Ashok Kumar(R151690)

ACKNOWLEDGEMENT

We would like to express my sincere gratitude to **Mr.Venkateshwarulu Naik** , our project internal guide for giving valuable suggestions and shown keen interest throughout the progress of our project.We are grateful to , HOD ECE, for providing excellent computing facilities and congenial atmosphere for progressing with my project. At the outset, I would like to thank Rajiv Gandhi University of Knowledge Technologies, R.K Valley for providing all the necessary resources for the successful completion of our project.

With sincere regards,

D.Rupesh Babu(R151737)

M.Ashok Kumar(R151690)

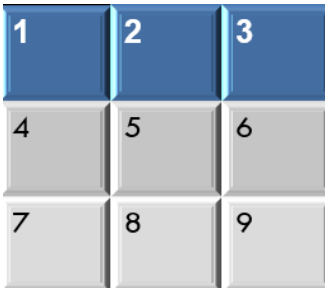
Contents:

1.Abstract:

This Project is aimed and designed to play Tic-Tac-Toe game on Monitor by using the Verilog code implemented on Basys3 Artrix7 board.Tic-Tac-Toe game is played on 3x3 grid cells which will be shown on a Monitor.For interface between User and Computer,a input device like Keyboard can be used.The entire game is circuited in Basys3 Artrix7 Board using Verilog Code.

2.Introduction:

- The Game can be played between two players.One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The players control their moves and marking using the KeyBoard.A Player should mark the desired cell with the symbol given in any one of unmarked cell in 9 cells.
- Then,the chance of marking goes to other player.This will be continued until noplace is left for marking or anyone of player successfully places the markings in the predefined sequence pair.
- The Game can be played between two players.One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.
- The players control their moves and marking using the KeyBoard.
- A Player should mark the desired cell with the symbol given in any one of unmarked cell in 9 cells.
- Then,the chance of marking goes to other player.This will be continued until noplace is left for marking or anyone of player successfully places the markings in the predefined sequence pair.
- If we consider the 3x3 grid on monitor as below:



- The player is considered to be win when he/she places the symbols like in this following pairs:
  (1,2,3) ; (4,5,6) ; (7,8,9) ; (1,4,7) ; (2,5,8) ; (3,6,9) ; (1,5,9) ; (3,5,7).

3.Hardware Description:

3.1.Basys3 Artix-7 FPGA Board:

The Basys 3board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7 Field Programmable Gate Array (FPGA) from Xilinx. With its high-capacity FPGA (Xilinx part number XC7A35T-1CPG236C),low overall cost, and collection of USB, VGA, and other ports, the Basys 3can host designs ranging from introductory combinational circuits to complex sequential circuits like embedded processors and controllers. It includes enough switches, LEDs,and other I/O devices to allow a large number of designs to be completed without the need for any additional hardware, and enough uncommitted FPGA I/O pins to allow designs to be expanded using Digilent Pmods or other custom boards and circuits.The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance,and more resources than earlier designs. Artix-7 35T features include:

.•33,280 logic cells in5200 slices (each slice contains four6-input LUTs and 8 flip-flops)

•1,800 Kbits of fast block RAM

•Five clock management tiles

•90 DSP slices

•Internal clock speeds exceeding 450MHz

•On-chip analog-to-digital converter (XADC

The Basys 3 also offers an improved collection of ports and peripherals, including:

•16 user switches

•16 user LEDs

•5 user pushbuttons

•4-digit 7-segment display

•Three Pmod ports

•Pmod for XADC signals

- 12-bit VGA output
- USB-UART Bridge
- Serial Flash
- Digilent USB-JTAG port for FPGA programming and communication

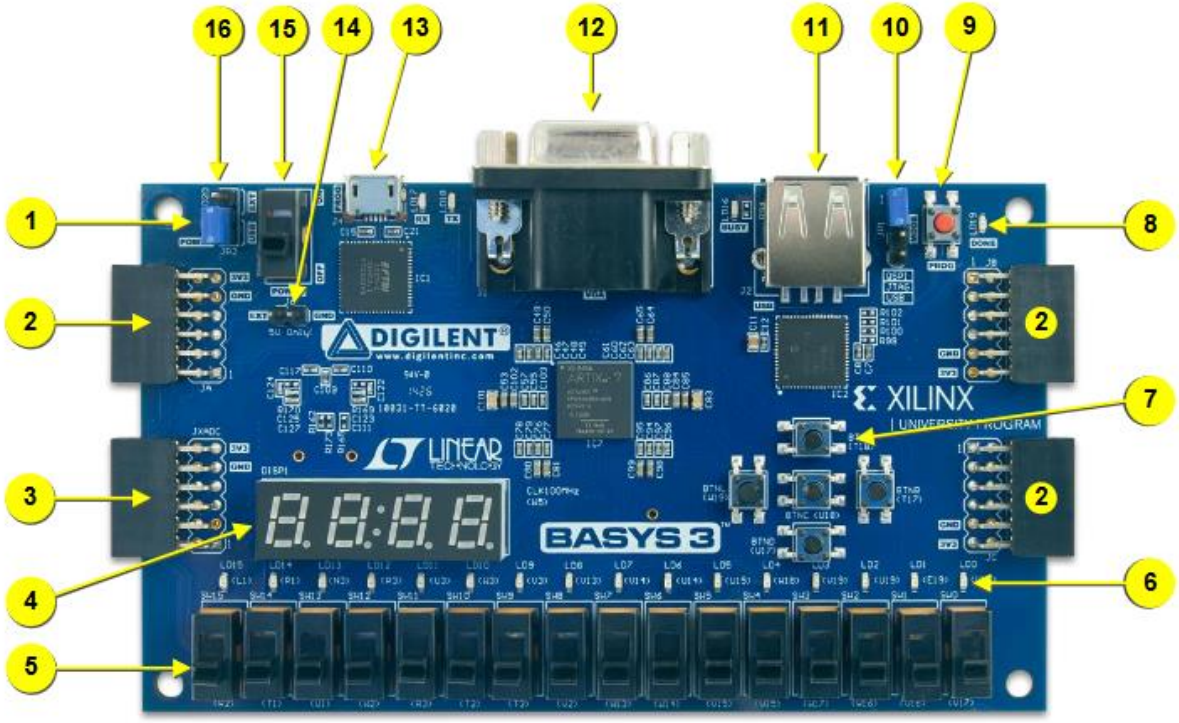


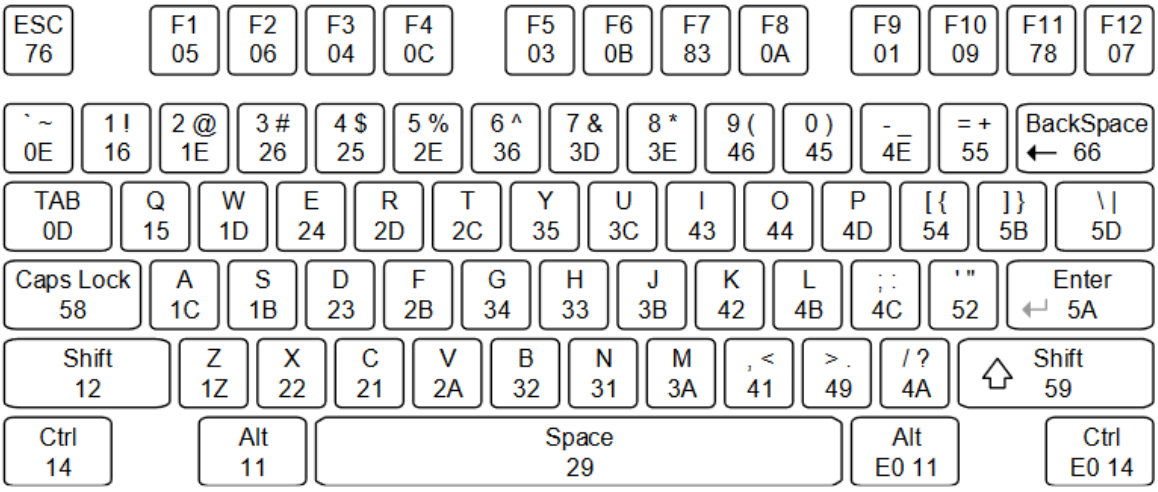
Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Table 1. Basys 3 Callouts and component descriptions.

3.2.Key Board:

Keyboard is used to take input from the user while playing the game.The keyboard uses open-collector drivers so the keyboard,or an attached host device,can drive the two-wire bus (if the host device will not send data to the keyboard, then the host can use input-only ports).PS/2-style keyboards use scan codes to communicate key press data. Each key is assigned a code that is sent whenever the key is pressed.If the key is held down, the scan code will be sent repeatedly about once every 100ms. When a key is released, anF0 key-up code is sent, followed by the scan code of the released key. If a key can be shifted to produce a new character (like a capital letter), then a shift character is sent in addition to the scan code and the host must determine which ASCII character to use. Some keys, called extended keys, send an E0 ahead of the scan code (and they may send more than one scan code). When an extended key is released, an E0 F0 key-up code is sent, followed by the scan code. Scan codes for most keys are shown in fig.



A host device can also send data to the keyboard. Table 3 shows a list of some common commands a host might send.The keyboard can send data to the host only when both the data and clock lines are high (or idle). Becausethe host is the bus master, the keyboard must check to see whether the host is sending data before driving the bus. To facilitate this, the clock line is used as a "clear to send"signal. If the host drivesthe clock line low, the keyboard must not send any data until the clock is released. The keyboard sends data to the host in 11-bit words that contain a '0'start bit, followed by 8-bits of scan code (LSB first),

followed by an odd parity bit,and terminated with a '1'stop bit.

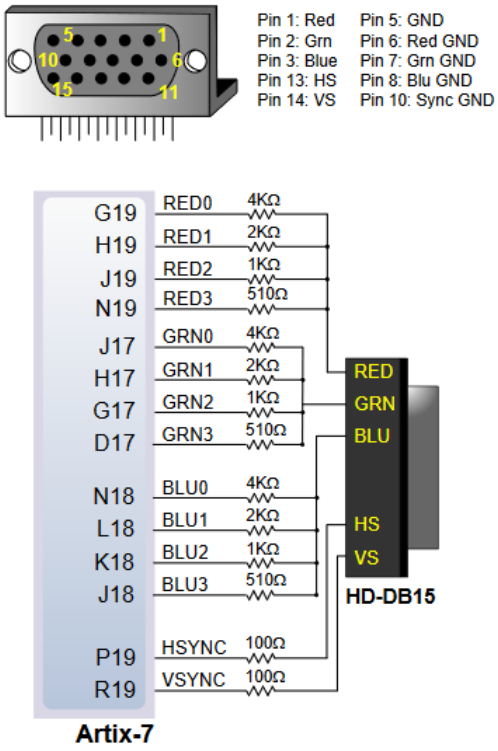
Command	Action
ED	Set Num Lock, Caps Lock, and Scroll Lock LEDs. Keyboard returns FA after receiving ED, then host sends a byte to set LED status: bit 0 sets Scroll Lock, bit 1 sets Num Lock, and bit 2 sets Caps lock. Bits 3 to 7 are ignored.
EE	Echo (test). Keyboard returns EE after receiving EE.
F3	Set scan code repeat rate. Keyboard returns F3 on receiving FA, then host sends second byte to set the repeat rate.
FE	Resend. FE directs keyboard to re-send most recent scan code.
FF	Reset. Resets the keyboard.

Table 3. Keyboard commands.

3.3.VGA Port

VGA is used to display the status of the game through a monitor.

The Basys 3board uses 14 FPGA signals to create a VGA port with 4-bits per color and the two standard sync signals (HS –Horizontal Sync, andVS –Vertical Sync). The color signals use resistor-divider circuits that work in conjunction with the 75 ohm termination resistance of the VGA display to create 16 signal levels each on the red, green, and blue VGA signals. This circuit, shown in Fig. 11, producesvideo color signals that proceed in equal increments between 0V (fully off) and 0.7V (fully on). Using this circuit, 4096 different colors can be displayed, one for each unique 12-bit pattern. A video controller circuit must be created in the FPGA to drive the sync and color signals with the correct timing in order to produce a working display system.CommandActionEASet stream mode. The mouse responds with "acknowledge" (0xFA) then resets its movement counters and enters stream mode.F4Enable data reporting. The mouse responds with "acknowledge" (0xFA) then enables data reporting and resets its movement counters. This command only affects behavior in stream mode. Once issued, mouse movement will automatically generate a data packet.F5Disable data reporting. The mouse responds with "acknowledge" (0xFA) then disables data reporting and resets its movement counters.F3Set mouse sample rate. The mouse responds with "acknowledge" (0xFA) then reads one more byte from the host. This byte is then saved as the new sample rate, and a new "acknowledge" packet is issued.FEResend. FE directs mouse to re-send last packet.FFReset. The mouse responds with "acknowledge" (0xFA) then enters reset mode.



3.4.VGA System Timing

CRT-based VGA displays use amplitude-modulated moving electron beams (or cathode rays) to display information on a phosphor-coated screen. LCD displays use an array of switches that can impose a voltage across a small amount of liquid crystal, thereby changing light permittivity through the crystal on a pixel-by-pixel basis. Although the following description is limited to CRT displays, LCD displays have evolved to use the same signal timingsas CRT displays (so the "signals"discussion below pertains to both CRTs and LCDs). Color CRT displays use three electron beams (one for red, one for blue, and one for green) to energize the phosphor that coats the inner side of the display end of a cathode ray tube.

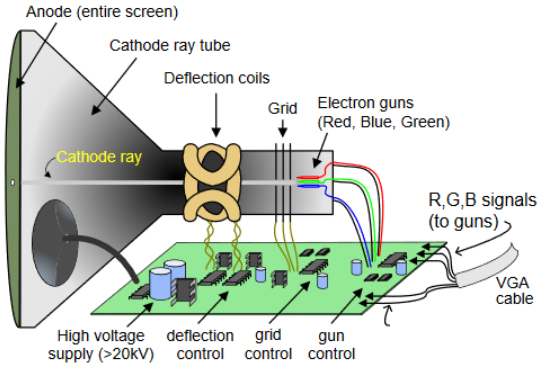


Figure 12. Color CRT display.

Electron beams emanate from "electron guns"which are finely-pointed heated cathodes placed in close proximity to a positively charged annular plate called a "grid."The electrostatic force imposed by the grid pulls rays of energized electrons from the cathodes, and those rays are fed by the current that flows into the cathodes. These particle rays are initially accelerated towards the grid, but they soon fall under the influence of the much larger electrostatic force that results from the entire phosphor-coated display surface of the CRT being charged to 20kV (or more). The rays are focused to a fine beam as they pass through the center of the grids, and then they accelerate to impact on the phosphor-coated display surface. The phosphor surface glows brightly at the impact point, and it continues to glow for several hundred microseconds after the beam is removed. The larger the current fed into the cathode, the brighter the phosphor will glow.Between the grid and the display surface, the beam passes through the neck of the CRT where two coils of wire produce orthogonal electromagnetic fields. Because cathode rays are composed of charged particles (electrons), they can be deflected by these magnetic fields. Current waveforms are passed through the coils to produce magnetic fields that interact with the cathode rays and cause them to transverse the display surface in a "raster"pattern, horizontally from left to right andvertically from top to bottom, as shown in Fig. 13. As the cathode ray moves over the surface of the display, the current sent to the electron guns can be increased or decreased to change the brightness of the display at the cathode ray impact point.Information is only displayed when the beam is moving in the "forward"direction (left to right and top to bottom), and not during the time the beam is reset back to the left or top edge of the display. Much of the potential display time is therefore lost in "blanking"periods when the beam is reset and stabilized to begin a new horizontal or vertical display pass. The size of the beams, the frequency at which the beam can be traced across the display, and the frequency at which the electron beam can be modulated determine the display resolution. Modern VGA displays can accommodate different resolutions, and a VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. The controller must produce synchronizing pulses at 3.3V (or 5V) to set the frequency at which current flows through the deflection coils, and it must ensure that video data is applied to the electron guns at the correct time. Raster video displays define a number of "rows"that corresponds to the number of horizontal passes the cathode makes over the display area, and a number of "columns"that corresponds to an area on each row that is assigned to one "picture element"or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns.The overall size of a display and the number of rows and columns determines the size of each pixel.

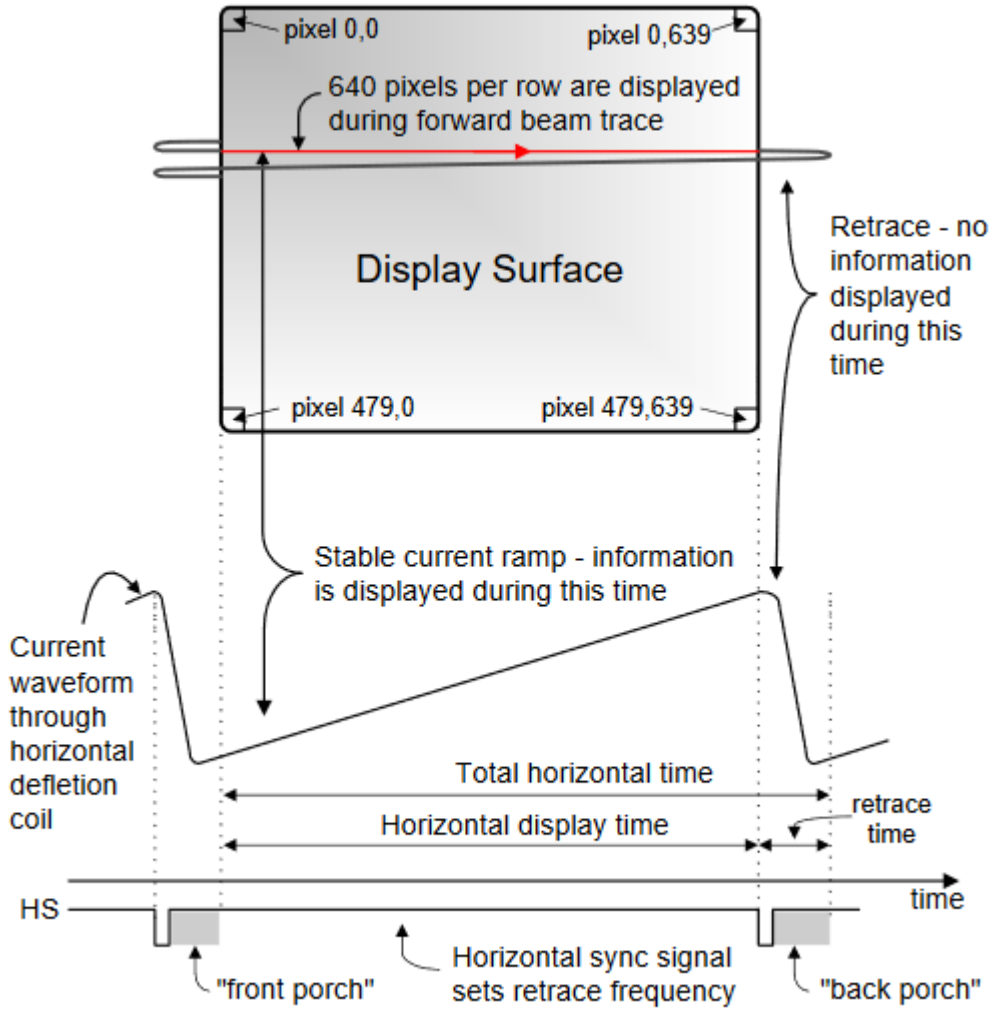


Figure 13. VGA horizontal synchronization.

Video data typically comes from a video refresh memory;with one or more bytes assigned to each pixel location (the Basys 3uses 12-bits per pixel). The controller must index into video memory as the beams move across the display, and retrieve and apply video data to the display at precisely the time the electron beam is moving across agiven pixel.A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the"refresh"frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display's phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal "retrace"frequency.

4.Software Description:

4.1.Vivado Design Suite

The Basys 3works with Xilinx's new high-performance Vivado Design Suite. Vivado includes many new tools and design flows that facilitate and enhance the latest design methods. It runs faster, allows better use of FPGA resources, and allows designers to focus their time evaluating design alternatives.

The FPGA enables the functionality of the chip to be programmed in, enabling this to be updated at any point required. This can be changed to accommodate updates or to even change the functionality a board or system when it is required to perform different functions.

The very name of the FPGA states that it is programmable. It is necessary to code to programme the FPGA. Knowing how to program an FPGA is a key skill and it forms a specialised area of electronic design.

4.2.FPGA programming approaches

There are several ways to develop the code to program an FPGA. In the very early days of FPGAs it might just have been possible to program the simplest FPGAs manually. Today this is not an option and a software program is required. There are several options open to FPGA developers:

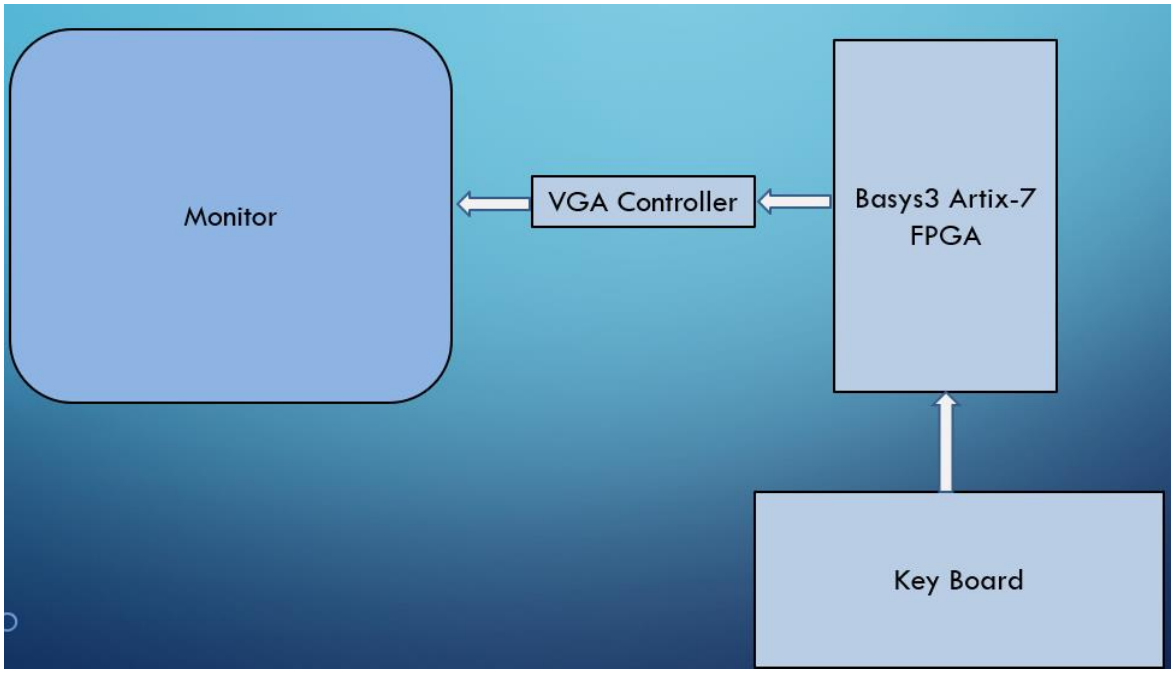
- ***VHDL:*** VHDL stands for VHSIC Hardware Description Language, where the VHSIC itself stands for Very High Speed Integrated Circuit. This FPGA programming language was developed by the US Department of Defense to document the behaviour of ASICs, or Application Specific Integrated Circuits. Base heavily on the programming language Ada, VHDL is a text language which has been very successful and popular for many years in programming FPGAs.
- ***Verilog:*** Verilog was the first form of hardware description language to be developed. It is standardised as IEEE 1364.
- ***LabVIEW FPGA:*** LabVIEW FPGA utilises the basic LabVIEW graphical interface but employs additional tools to enable it to provide the functionality required for programming FPGAs.

In this Project,Verilog HDL is used:

Verilog is a HARDWARE DESCRIPTION LANGUAGE (HDL). It is a language used for describing a digital system like a network switch or a

microprocessor or a memory or a flip−flop. It means, by using a HDL we can describe any digital hardware at any level. Designs, which are described in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

5.Block Diagram:



Monitor:Used to display the status of the Game.

VGA Controller: Connection between Monitor and Basys3 which is used to control the Pixels of the monitor.

Key Board: Used to give inputs to the Basys3.It is the interaction between Players and the Board.

6.Verilog Code:

```
module tic_tac_toe_game(
     input clock, // clock of the game
     input reset, // reset button to reset the game
     input play, // play button to enable player to play
     input pc, // pc button to enable computer to play
     input [3:0] computer_position,player_position,
     // positions to play
     output wire [1:0] pos1,pos2,pos3,
     pos4,pos5,pos6,pos7,pos8,pos9,// LED display for
positions
     output wire[1:0]who      );
 wire [15:0] PC_en;// Computer enable signals
 wire [15:0] PL_en; // Player enable signals
 wire illegal_move;
 wire win; // win signal
 wire computer_play; // computer enabling signal
 wire player_play; // player enabling signal
 wire no_space; // no space signal
  position_registers position_reg_unit(
      clock, // clock of the game
      reset, // reset the game
      illegal_move,
      PC_en[8:0], // Computer enable signals
      PL_en[8:0], // Player enable signals
      pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
      );
winner_detector
win_detect_unit(pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,w
in,who);
 position_decoder pd1(computer_position,computer_play,PC_en);
position_decoder pd2(player_position,player_play,PL_en);
  illegal_move_detector imd_unit(
   pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   PC_en[8:0], PL_en[8:0],
   illegal_move
   );
nospace_detector nsd_unit(
   pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
```

```
    no_space
     );
 fsm_controller tic_tac_toe_controller(
      clock,
      reset,
      play,
      pc,
      illegal_move,
      no_space, // no_space detected
      win, // winner detected
      computer_play,
      player_play
      );
endmodule
module position_registers(
       input clock,
       input reset,
       input illegal_move,
       input [8:0] PC_en,
       input [8:0] PL_en,
       output reg[1:0]
pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9
       );
always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos1 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos1 <= pos1;// keep previous position
   else if(PC_en[0]==1'b1)
    pos1 <= 2'b10; // store computer data
   else if (PL_en[0]==1'b1)
    pos1 <= 2'b01;// store player data
   else
    pos1 <= pos1;// keep previous position
  end
 end
 always @(posedge clock or posedge reset)
 begin
  if(reset)
```

```
    pos2 <= 2'b00;
   else begin
    if(illegal_move==1'b1)
     pos2 <= pos2;
    else if(PC_en[1]==1'b1)
     pos2 <= 2'b10;
    else if (PL_en[1]==1'b1)
     pos2 <= 2'b01;
    else
     pos2 <= pos2;  end
  end
always @(posedge clock or posedge reset)
  begin
   if(reset)
    pos3 <= 2'b00;
   else begin
    if(illegal_move==1'b1)
     pos3 <= pos3;
    else if(PC_en[2]==1'b1)
     pos3 <= 2'b10;
    else if (PL_en[2]==1'b1)
     pos3 <= 2'b01;
    else
     pos3 <= pos3;
   end
  end
always @(posedge clock or posedge reset)
  begin
   if(reset)
    pos4 <= 2'b00;
   else begin
    if(illegal_move==1'b1)
     pos4 <= pos4;
    else if(PC_en[3]==1'b1)
     pos4 <= 2'b10;
    else if (PL_en[3]==1'b1)
     pos4 <= 2'b01;
    else
     pos4 <= pos4;
   end
  end
```

```
 always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos5 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos5 <= pos5;
   else if(PC_en[4]==1'b1)
    pos5 <= 2'b10;
   else if (PL_en[4]==1'b1)
    pos5 <= 2'b01;
   else
    pos5 <= pos5;
  end
 end
always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos6 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos6 <= pos6;
   else if(PC_en[5]==1'b1)
    pos6 <= 2'b10;
   else if (PL_en[5]==1'b1)
    pos6 <= 2'b01;
   else
    pos6 <= pos6;
  end
 end
always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos7 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos7 <= pos7;
   else if(PC_en[6]==1'b1)
    pos7 <= 2'b10;
   else if (PL_en[6]==1'b1)
    pos7 <= 2'b01;
```

```
    else
     pos7 <= pos7;
   end
  end
always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos8 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos8 <= pos8;
   else if(PC_en[7]==1'b1)
    pos8 <= 2'b10;
   else if (PL_en[7]==1'b1)
    pos8 <= 2'b01;
   else
    pos8 <= pos8;
  end
 end
 always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos9 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos9 <= pos9;
   else if(PC_en[8]==1'b1)
    pos9 <= 2'b10;
   else if (PL_en[8]==1'b1)
    pos9 <= 2'b01;
   else
    pos9 <= pos9;
  end
 end
endmodule
module fsm_controller(
     input clock,// clock of the circuit
     input reset,
     play, // player plays
     pc,// computer plays
     illegal_move,// illegal move detected
```

```
      no_space, // no_space
      win, // winner detected
      output reg computer_play, // enable computer to play
      player_play);
parameter IDLE=2'b00;
parameter PLAYER=2'b01;
parameter COMPUTER=2'b10;
parameter GAME_DONE=2'b11;
reg[1:0] current_state, next_state;
always @(posedge clock or posedge reset)
begin
 if(reset)
  current_state <= IDLE;
 else
  current_state <= next_state;
end
 // next state
always @(*)
 begin
 case(current_state)
 IDLE: begin
  if(reset==1'b0 && play == 1'b1)
   next_state <= PLAYER; // player to play
  else
   next_state <= IDLE;
  player_play <= 1'b0;
  computer_play <= 1'b0;
 end
 PLAYER:begin
  player_play <= 1'b1;
  computer_play <= 1'b0;
  if(illegal_move==1'b0)
   next_state <= COMPUTER; // computer to play
  else
   next_state <= IDLE;
 end
 COMPUTER:begin
  player_play <= 1'b0;
  if(pc==1'b0) begin
   next_state <= COMPUTER;
   computer_play <= 1'b0;
```

```
  end
  else if(win==1'b0 && no_space == 1'b0)
  begin
   next_state <= IDLE;
   computer_play <= 1'b1;// computer to play when PC=1
  end
  else if(no_space == 1 || win ==1'b1)
  begin
   next_state <= GAME_DONE; // game done
   computer_play <= 1'b1;// computer to play when PC=1
  end
 end
 GAME_DONE:begin // game done
  player_play <= 1'b0;
  computer_play <= 1'b0;
  if(reset==1'b1)
   next_state <= IDLE;// reset the game to IDLE
  else
   next_state <= GAME_DONE;
 end
 default: next_state <= IDLE;
 endcase
 end
endmodule
module nospace_detector(
   input [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   output wire no_space
    );
wire temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9;
assign temp1 = pos1[1] | pos1[0];
assign temp2 = pos2[1] | pos2[0];
assign temp3 = pos3[1] | pos3[0];
assign temp4 = pos4[1] | pos4[0];
assign temp5 = pos5[1] | pos5[0];
assign temp6 = pos6[1] | pos6[0];
assign temp7 = pos7[1] | pos7[0];
assign temp8 = pos8[1] | pos8[0];
assign temp9 = pos9[1] | pos9[0];
// output
assign no_space =(((((((temp1 & temp2) & temp3) & temp4) &
temp5) & temp6) & temp7) & temp8) & temp9);
```

```
endmodule
module illegal_move_detector(
   input [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   input [8:0] PC_en, PL_en,
   output wire illegal_move);
wire temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9;
wire
temp11,temp12,temp13,temp14,temp15,temp16,temp17,temp18,temp19
;
wire temp21,temp22;
assign temp1 = (pos1[1] | pos1[0]) & PL_en[0];
assign temp2 = (pos2[1] | pos2[0]) & PL_en[1];
assign temp3 = (pos3[1] | pos3[0]) & PL_en[2];
assign temp4 = (pos4[1] | pos4[0]) & PL_en[3];
assign temp5 = (pos5[1] | pos5[0]) & PL_en[4];
assign temp6 = (pos6[1] | pos6[0]) & PL_en[5];
assign temp7 = (pos7[1] | pos7[0]) & PL_en[6];
assign temp8 = (pos8[1] | pos8[0]) & PL_en[7];
assign temp9 = (pos9[1] | pos9[0]) & PL_en[8];
// computer : illegal moving
assign temp11 = (pos1[1] | pos1[0]) & PC_en[0];
assign temp12 = (pos2[1] | pos2[0]) & PC_en[1];
assign temp13 = (pos3[1] | pos3[0]) & PC_en[2];
assign temp14 = (pos4[1] | pos4[0]) & PC_en[3];
assign temp15 = (pos5[1] | pos5[0]) & PC_en[4];
assign temp16 = (pos6[1] | pos6[0]) & PC_en[5];
assign temp17 = (pos7[1] | pos7[0]) & PC_en[6];
assign temp18 = (pos8[1] | pos8[0]) & PC_en[7];
assign temp19 = (pos9[1] | pos9[0]) & PC_en[8];
assign temp21 =(((((((temp1 | temp2) | temp3) | temp4) |
temp5) | temp6) | temp7) | temp8) | temp9);
assign temp22 =(((((((temp11 | temp12) | temp13) | temp14) |
temp15) | temp16) | temp17) | temp18) | temp19);
// output illegal move
assign illegal_move = temp21 | temp22 ;
endmodule
module position_decoder(input[3:0] in, input enable, output
wire [15:0] out_en);
 reg[15:0] temp1;
 assign out_en = (enable==1'b1)?temp1:16'd0;
 always @(*)
```

```
 begin
 case(in)
 4'd0: temp1 <= 16'b0000000000000001;
 4'd1: temp1 <= 16'b0000000000000010;
 4'd2: temp1 <= 16'b0000000000000100;
 4'd3: temp1 <= 16'b0000000000001000;
 4'd4: temp1 <= 16'b0000000000010000;
 4'd5: temp1 <= 16'b0000000000100000;
 4'd6: temp1 <= 16'b0000000001000000;
 4'd7: temp1 <= 16'b0000000010000000;
 4'd8: temp1 <= 16'b0000000100000000;
 4'd9: temp1 <= 16'b0000001000000000;
 4'd10: temp1 <= 16'b0000010000000000;
 4'd11: temp1 <= 16'b0000100000000000;
 4'd12: temp1 <= 16'b0001000000000000;
 4'd13: temp1 <= 16'b0010000000000000;
 4'd14: temp1 <= 16'b0100000000000000;
 4'd15: temp1 <= 16'b1000000000000000;
 default: temp1 <= 16'b0000000000000001;
 endcase
end
endmodule
module winner_detector(input [1:0]
pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9, output wire
winner, output wire [1:0]who);
wire win1,win2,win3,win4,win5,win6,win7,win8;
wire [1:0] who1,who2,who3,who4,who5,who6,who7,who8;
winner_detect_3 u1(pos1,pos2,pos3,win1,who1);// (1,2,3);
winner_detect_3 u2(pos4,pos5,pos6,win2,who2);// (4,5,6);
winner_detect_3 u3(pos7,pos8,pos9,win3,who3);// (7,8,9);
winner_detect_3 u4(pos1,pos4,pos7,win4,who4);// (1,4,7);
winner_detect_3 u5(pos2,pos5,pos8,win5,who5);// (2,5,8);
winner_detect_3 u6(pos3,pos6,pos9,win6,who6);// (3,6,9);
winner_detect_3 u7(pos1,pos5,pos9,win7,who7);// (1,5,9);
winner_detect_3 u8(pos3,pos5,pos6,win8,who8);// (3,5,7);
assign winner = ((((((win1 | win2) | win3) | win4) | win5) |
win6) | win7) | win8);
assign who = ((((((who1 | who2) | who3) | who4) | who5) |
who6) | who7) | who8);
endmodule
```

```
module winner_detect_3(input [1:0] pos0,pos1,pos2, output wire
winner, output wire [1:0]who);
wire [1:0] temp0,temp1,temp2;
wire temp3;
assign temp0[1] = !(pos0[1]^pos1[1]);
assign temp0[0] = !(pos0[0]^pos1[0]);
assign temp1[1] = !(pos2[1]^pos1[1]);
assign temp1[0] = !(pos2[0]^pos1[0]);
assign temp2[1] = temp0[1] & temp1[1];
assign temp2[0] = temp0[0] & temp1[0];
assign temp3 = pos0[1] | pos0[0];
// winner if 3 positions are similar and should be 01 or 10
assign winner = temp3 & temp2[1] & temp2[0];
assign who[1] = winner & pos0[1];
assign who[0] = winner & pos0[0];
endmodule
```

7.Test Bench:

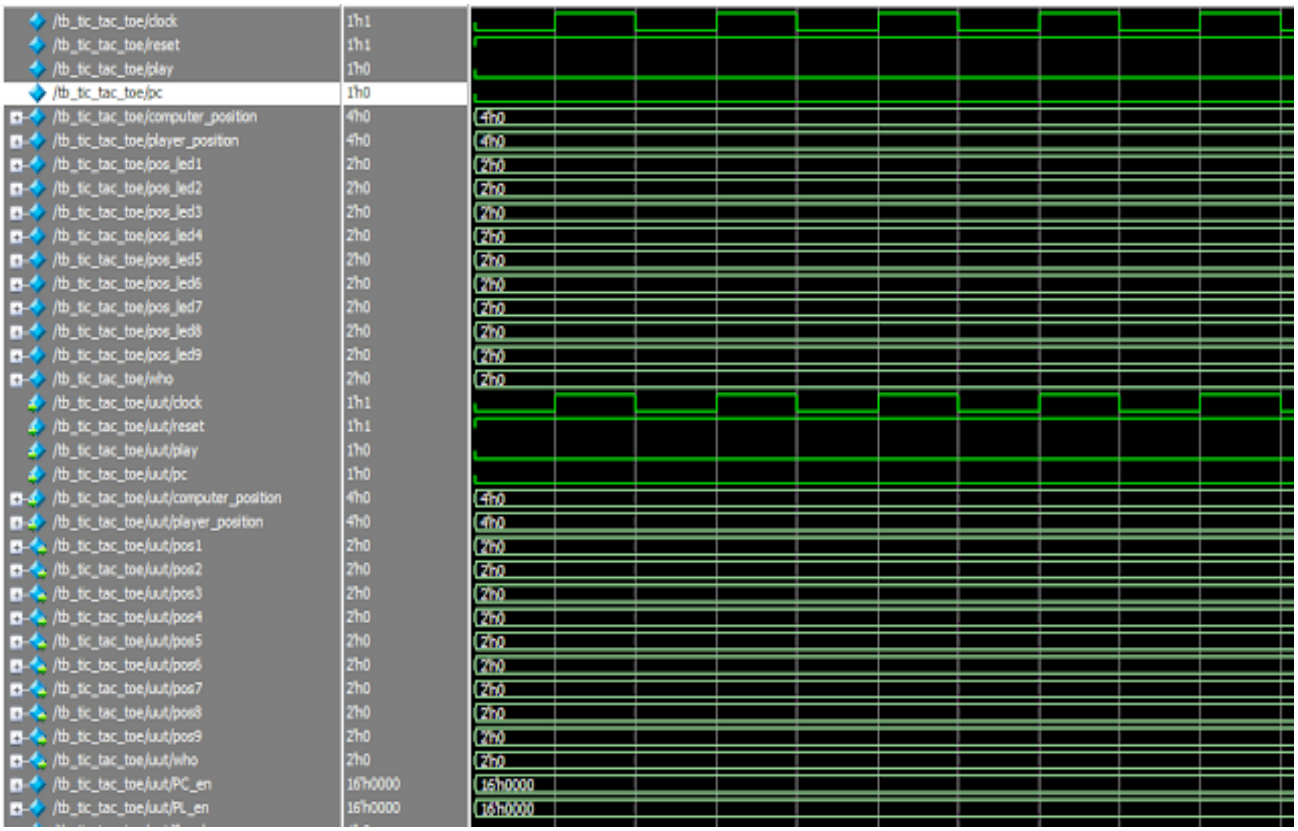
```
`timescale 1ns / 1ps
module tb_tic_tac_toe;
 // Inputs
 reg clock;
 reg reset;
 reg play;
 reg pc;
 reg [3:0] computer_position;
 reg [3:0] player_position;
 // Outputs
 wire [1:0] pos_led1;
 wire [1:0] pos_led2;
 wire [1:0] pos_led3;
 wire [1:0] pos_led4;
 wire [1:0] pos_led5;
 wire [1:0] pos_led6;
 wire [1:0] pos_led7;
 wire [1:0] pos_led8;
 wire [1:0] pos_led9;
 wire [1:0] who;
 tic_tac_toe_game uut (
  .clock(clock),
```

```
  .reset(reset),
  .play(play),
  .pc(pc),
  .computer_position(computer_position),
  .player_position(player_position),
  .pos1(pos_led1),
  .pos2(pos_led2),
  .pos3(pos_led3),
  .pos4(pos_led4),
  .pos5(pos_led5),
  .pos6(pos_led6),
  .pos7(pos_led7),
  .pos8(pos_led8),
  .pos9(pos_led9),
  .who(who)
 );
initial begin
 clock = 0;
 forever #5 clock = ~clock;
 end
 initial begin
  // Initialize Inputs
  play = 0;
  reset = 1;
  computer_position = 0;
  player_position = 0;
  pc = 0;
  #100;
  reset = 0;
  #100;
  play = 1;
  pc = 0;
  computer_position = 4;
  player_position = 0;
  #50;
  pc = 1;
  play = 0;
  #100;
  reset = 0;
  play = 1;
  pc = 0;
```

```
    computer_position = 8;
    player_position = 1;
    #50;
    pc = 1;
    play = 0;
    #100;
    reset = 0;
    play = 1;
    pc = 0;
    computer_position = 6;
    player_position = 2;
    #50;
    pc = 1;
    play = 0;
    #50
    pc = 0;
    play = 0;
    end

endmodule
```

8.Timing Diagram:



9.Conclusion:

- We can design any complex electronic designs using the FPGA even Microcontrollers and Microprocessors. This Project shows us how it can control the monitor pixels by using the game Tic-Tac-Toe.
- It is more popular due to the fact that it can be reprogrammable unlike ASIC which can’t be reprogrammable.So,it can serve for any kind of application over and over again just by configuring the Code.

10.References:

1.https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf

2.https://reference.digilentinc.com/_media/basys3:basys3_rm.pdf

3. http://www.iosrjournals.org/iosr-jvlsi/papers/vol6-issue6/Version-2/J0606028286.pdf

4. https://arxiv.org/ftp/arxiv/papers/1406/1406.5177.pdf